ROACH

MULTICS TECHNICAL BULLETIN MTB-329

To: Distribution

From: A. Bensoussan

Date: 03/21/77

Subject: A NEW PROPOSAL FOR ACCESSING LARGE FILES

INTRODUCTION

This document explains why the Multi-Segment approach to large files implementation is inefficient and proposes an alternative that would eliminate the basic causes of the current overhead. The new proposal calls for a reorganization of the file structure and a new programming protocol for accessing a file.

This document deals with what a file would look like to a programmer who would write vfile-like programs, and how he could access it. It also gives the list of the new ring zero primitives that would be made available to him, with a description of what they do for him. The description of how they do it will be given in a subsequent document.

The following topics are discussed:

- o Current organization and access of large files
- o Overhead of the current method
- o Bases of the new solution
- o Pathname of a file
- o Logical to physical address mapping
- o Volume assignment and quota
- o Access Control
- o File accessing protocol
- o Advantages of the new method
- o File Control Module primitives
- o Disk organization
- o Coexistence of files and segments in one volume
- o Coexistence of old files and new files

CURRENT ORGANIZATION AND ACCESS OF LARGE FILES

In the current Multics System, files larger than 256K words are organized into "multi-segment files" (MSF). By convention, a file whose pathname is F is represented by the directory with the pathname F and a collection of 256K segments with the pathnames F>0, F>1, F>2...F>n. The file can be viewed as being the juxtaposition of all these 256K segments in the order of their increasing entry names.

A word in a file is uniquely specified by the component number (i.e. the entry name of the 256K segment in which it resides) and its relative word number within this component. In order to access a word whose address is (compno, wordno) it is necessary to initiate the segment whose pathname is F>compno. Let segno be the segment number assigned to the component. The reference (segno, wordno) will cause the component to be activated (if not already active), that is, a 256-entry page table is allocated and initialized with the disk addresses read from the VTOCE entry of that component.

OVERHEAD OF THE CURRENT METHOD

The method just described is not particularly efficient for very large files, i.e. files that would be made of hundreds or thousands of segments, especially when these files are accessed randomly. I believe two basic reasons to be the cause of this overhead:

1. Each time a page which does not have a PTW in core is referenced, a lot of energy is spent to activate the entire 256K segment in which the referenced page resides, providing a PTW for 256 pages with the assumption that they have a good chance to be needed soon since they are part of the same segment. Actually, when large files are accessed randomly, the probability for another page of the same segment to be referenced while the segment is still active is almost zero. And the time, space and I/O spent to activate the not-needed 255 pages is wasted.

2. A large amount of data is used to map the logical address within a file into a physical address, causing additional page faults and processing. Included here are: the directory for the MSF, all branches and VTOC entries needed to describe the various components of the file, all KST entries and all SDW's for those components that happen to be initiated. Although some may argue that branches do not contain any physical attributes, it is clear that the 256K components of a file do not represent any logical entity with respect to the file. Their entry names are sequential numbers, their access rights are identical and their size has been set by convention. The only purpose of this

organization into MSF is to allow the virtual memory manager to take care of the mapping between (component number, word number) and the physical address.

The MSF technique is definitely attractive in the sense that it requires no additional mechanism. It uses the entire existing Multics virtual memory apparatus: directory control, segment control, access control, vtoc manager, page control with its disk allocation mechanism and quota checking, back-up system and salvager. Unfortunately, it is too expensive, particularly when files are very large and are accessed randomly; this is why a different approach is considered as an alternative to the current MSF implementation.

THE BASES OF THE NEW SOLUTION

The new solution attempts to eliminate or minimize the overhead due to each of the 2 causes mentioned above.

1. With the strong belief that the most common way to access large files is randomly, we want to optimize access to a single page. Each reference to a page which is not in core will cause the fault handler to do just what is necessary to make this particular page accessible to the process, without concern for the preceding or following pages.

   In the most common case one will arrange for a given page to be accessed by the hardware through a one-word page table, as if it were a one-page segment.

   In some special cases, when the user knows that he has to make random access within a set of n logically contiguous pages of the file, one will arrange for these n contiguous pages to be accessed by the hardware through an n-word page table as if these pages were an n-page segment ($n \leq 256$ of course).

   It is clear that if a process is to reference randomly every single page of a file, it will not be possible to provide for a one to one mapping between segment numbers and pages of the file since the hardware does not support such a large size for the descriptor segment. The (vfile) programmer will have to multiplex segment numbers among pages and remember the logical page number(s) of those pages described by a given segment number at a specific time.

2. A file is no longer organized into segments. It is merely a collection of logically contiguous pages. There is no longer a directory for the file, no longer a branch and a VTOCE for each component. Furthermore, disk records are no longer allocated one at a time but a group (hopefully large) at a time, so that for a set of logically contiguous pages,

the disk addresses are also contiguous and can be calculated from the first one.

The rest of the document describes in more detail how a large file would be organized and accessed, and gives the list of the new ring zero primitives available to the user, with a description of what each primitive is supposed to do.

PATHNAME OF A FILE

The pathname of a file leads, in the directory hierarchy, to a ring zero segment referred to as the File Definition Segment. This ring zero segment could be regarded as a large branch. It is not implemented as a branch in a directory so that no change is required to directory control. The primary purpose of this ring zero segment is to provide the file map and the access control information for the file.

LOGICAL TO PHYSICAL ADDRESS MAPPING

The logical address that we are concerned with here is the page number. A file is a collection of logically contiguous 1024-word pages, numbered sequentially from 0 to N, N being limited only by the disk capacity. The physical address of a page consists of the pair (physical volume id, record number). It is expected that the file map may contain a small number of entries although the number of logical pages may be very large.

This will be achieved by organizing the file into sections, for which contiguous logical addresses will be mapped into contiguous disk addresses. The disk space is no longer allocated on a page by page basis, as it is in the current system, but by groups of contiguous disk records, these groups being as large as possible.

The number of records in a section can be as large as the number of records available in a single physical volume. It is left to the owner of the file or to the database manager to decide how many physical records should be allocated in a given section.

It is conceivable that users of large files may take advantage of that feature by making an entire disk a single section. By doing so, they minimize the time spent in allocation, they minimize the overhead to access the file since the file map is so small that it can be kept in core, and finally they may use it as a physical placement control since they can cause some data to reside in a particular location of the disk by giving it the appropriate logical address.

VOLUME ASSIGNMENT AND QUOTA

In which physical volume(s) should or can a file be stored and how is the quota managed?

One can keep the policy that the new storage system uses for segments. A file with the pathname DIR>F could be stored in any physical volume that is a member of the son's logical volume associated with the directory DIR. Allocation of disk records for this file is permitted provided it does not cause an overflow of the quota cell to which all pages of non-directory segments directly inferior to DIR are charged. Such a policy is homogeneous with what is done for segments and would not be difficult to implement.

However, one may wonder if this is a realistic policy for files that use 50 or 100 disk packs? Any comment or suggestions on that subject will be greatly appreciated.

ACCESS CONTROL

The same access control features as those currently available for MSF files will be available in the new proposal, with the same guarantee of being enforced.

What is available today is the standard ACL mechanism. All components of the same MSF have the same ring brackets and the same ACL.

In the new file organization the File Definition Segment is a ring zero segment, with read and write permission for every user, while operating in ring zero. This segment also has an extended ACL and extended ring brackets which represents the ACL and ring brackets of the file itself with respect to the users. Any SDW manufactured in a user process and describing any portion of a given file will have the ring brackets and access mode specified by the extended ring brackets and the extended mode specified for the user in the extended ACL associated with the ring zero File Definition Segment.

A mechanism to support immediate access revocation will be provided. It will be possible to invoke it unconditionally each time the ACL of the file is modified. However, my opinion is that it is not desirable to make this policy the standard policy, which no one could bypass. In many cases, the user who removes access to a file from another user is willing to let this other user terminate whatever he may currently be doing with the file at the time the ACL is modified. The reason being that more damage could be done to the file by immediately revoking access from somebody using it than by postponing the revocation.

I would suggest that both forms of ACL modification be available, one with immediate revocation and the other with revocation taking effect as soon as the user from whom access is revoked is through using the file.

FILE ACCESSING PROTOCOL

The (vfile) programmer will have to become familiar with a new protocol to access a file, and with a new kind of special segments which have no branch, no ACL, no VTOCE and that one will be referred to as "window segments".

The first step in the protocol is to call a ring 0 primitive to "initiate an N-page window segment for a file F and get back its segment number S". At this point, it is agreed between the users and the supervisor, that segment S will be used to describe any window of N contiguous pages in file F. However, segment number S cannot be used yet since the position of the window is still undefined.

The next step in the protocol is to call another ring zero primitive to "position the window segment S to a particular window starting at page number P of the file". At this point, a hardware reference using the address (S,i) in the user process will cause the word 1024*P + i of the file to be accessed, provided that $0 \leq i < 1024*N$.

The programmer can move the window segment to look at another portion of the file by calling again the ring zero primitive to "position the window segment S to page P1". At this point, segment S describes a new region of the file and the address (S,i) will cause the word 1024*P1 + i of the file to be accessed. Several window segments may be initiated for the same file. The programmer may position them at different regions of the file. He may keep some of them pointing at a given position if he knows he will need to access again this portion of the file, while he may reposition the others to various parts of the file. It is legitimate to have 2 different window segments describing the same region or overlapping regions of the file.

This new access method can be illustrated by showing how vfile would use it for large indexed files. Since the index is organized into individual pages, without any meaning for page sequentiality, it is an ideal candidate to be accessed by one page window segments. Thus, the vfile programmer would initiate a few (3 or 4) one-page window segments to access the index. He would position one of them to the page containing the root of the B-tree and would keep it positioned to that page while the file is opened. Since the root needs to be accessed for any index operation, he would position the others to the various pages of the index that has to be accessed during a given index operation. In addition, he could use an n-page window segment that he would

position to the record that must be copied from or to the user area, n being derived from the maximum or the average record size. Note that he could also use a one-page window segment that he would position to the page where the record starts and that he would reposition to the following page if the record crosses the page boundary and so on until he reaches the end of the record. The size of the window segment he should use will depend on the price to pay in CPU time for repositioning a window segment versus the price to pay in wired core memory for a large window segment.

ADVANTAGES OF THE NEW METHOD

The first advantage, of course, is that a greater efficiency can be expected for the reasons explained earlier. If we assume that, for random access to large files, most of the time a reference to a new page causes a segment fault in the current system, then it is fair to say that this segment fault will be replaced by a call to the position primitive in the new system. The position primitive will be much faster than a segment fault on a 256K segment because the following items do not need to be accessed since they do not even exist: the KST entry for the component, the branch of the component, the MSF directory, its lock during activation, the VTOC entry of the component, the VTOC entry of the component that has to be deactivated. In addition, the position primitive will use its own locks for synchronization while the segment fault causes more contention on the directory lock table, the AST lock, the VTOC manager lock, and the page control lock. Also, there is no need any longer to try to guess what is the best candidate for deactivation since the caller of the position primitive expresses very clearly that he is no longer interested in those pages described by the window segment. This information communciated by the user can also be valuable in making a better choice than page control for selecting file pages to be removed from core. And finally, all page faults on file pages will not be handled by page control any longer and therefore can be processed in parallel with the page fault handling for segments since a different lock will be used, to protect databases that are independent of those used by page control.

The second advantage, which has not yet been mentioned but which is very significant, is that this method scales up gracefully when files become very large, while in the MSF approach the size of the file may be limited by the number of entries that a directory can hold, or by the number of entries that the KST can hold or by the number of segments that the descriptor segment can hold. With the new method, growing a file is accomplished by adding a new "section", that is adding a new entry to the file map.

FILE CONTROL MODULE PRIMITIVES

The File Control Module consists of a set of new ring zero procedures and databases needed to support the new file access and organization. This paragraph gives the list and specification of those procedures that are available to the user as new ring zero gates.

- o Create_file (pathname)

  Creates the File Definition Segment with the specified pathname. This segment is a ring zero segment with rw *.*.* in the regular ACL. The extended ring brackets are val,val,val and the extended ACL has rw for the user of the current process. This segment is initialized with the appropriate initial value, as required by the implementation, showing that the file is empty, i.e. has no disk record allocated to it.

- o Open_file (pathname, fileno)

  The only purpose of this primitive is to return a file number (fileno) that the user must use to refer to the file in the other primitives.

- o Initiate_window_segment (fileno, npages, segno)

  Returns to the user a segment number "segno" that was not used yet, and that the user can now use to describe any "npages" contiguous pages of the file defined by "fileno". Initializes the SDW showing that the position of the file that it describes is yet undefined. The file number must designate a file which is currently open (with respect to the file control module) in this process. The number of pages must be between 1 and 256.

- o Position_window_segment (segno, pageno)

  Takes necessary action so that the window segment "segno" describes as many contiguous pages as it has been initiated for, in the file that it has been initiated for, starting at the page number "pageno" of the file. In addition, it

initializes the SDW for the window segment with the access rights the user has to this file.

- Allocate_disc (fileno, pageno, npages)

  Causes "npages" contiguous pages of the file defined by "fileno" and starting at page number "pageno" to be allocated contiguous disk addresses. The volume in which these disk records are allocated and the quota checking must conform to whatever policy will be chosen for large files.

- Free_disk (fileno, pageno, npage)

  Causes the disk records that were allocated to the "npages" consecutive pages of file "fileno", starting at the page "pageno" to be freed. Also causes any subsequent attempt to access these pages to fail.

- Close_file (fileno)

  Causes the file number "fileno" to be no longer associated with the file it currently defines, and all window segments initiated for this file to be terminated, i.e. their segment numbers become unused.

- Delete_file (pathname)

  Causes the file specified by the pathname to be deleted. That is, all disk records allocated to this file must be freed and the ring zero File Definition Segment deleted.

- Recompute_access (pathname)

  Causes all users that are currently using the file to have their access rights for this file recalculated in order to reflect the latest ACL modification.

DISK ORGANIZATION

A file is made of one or more sections. Each section corresponds to a set of contiguous logical addresses mapped into contiguous physical addresses. A section is entirely contained in the same disk. It is defined by the following items:

(pageno, number of pages, pvid, recordno)

A file may be made of any number of sections, in any number of physical volumes. All those sections that reside in the same volume are described by VTOC entries in this volume. A VTOCE can describe up to 64 sections; if the file has more than 64 sections in the same volume, additional VTOC entries will be used as needed to describe all the sections.

The file map of a file really begins in the ring zero File Definition Segment, where a list of VTOC entry pointers is found. Each of these VTOCE pointers points to a VTOCE that describes up to 64 sections of the file.

A VTOCE is expected to be 192 words long (like a VTOC entry for a segment) in such a way that it can be accessed using the current VTOC manager. The header may contain the same kind of information it does for a segment, whenever it is relevant. The file map portion of a VTOCE, instead of containing an array of 256 record numbers, will contain an array of 64 section descriptors (pageno, recordno, number of records), the pvid being the pvid of the volume in which the VTOCE resides. The trailer portion of the VTOCE may contain the same kind of information it has for a segment, and, in particular, may point back to the File Definition Segment.

The volume has a label, like any other volume, and also a volume map, which consists of one bit per record. It is possible, by reading the VTOC portion of a disk to determine what files are stored in the volume, and more precisely, what sections of what files are stored in the volume. It is also possible to remanufacture the volume map from the VTOC portion of the disk.

The general philosophy used in the new storage system whereby a disk must start by its table of contents, has been the dominant idea in defining how a volume containing files should be organized.

COEXISTENCE OF FILES AND SEGMENTS IN ONE VOLUME

One would like to be able to store files as well as segments on the same physical volume. The disk organization just described has been chosen in order to make the coexistence possible. It is clear that the major program that would be affected by this coexistence is the physical volume Salvager. Also page control and file control would have to be synchronized when allocating disk records.

It is possible however to have a first version of the system where files and segments are forced to be in different volumes so that no change to the Salvager and no change to page control would be needed. Such a version would be useful to demonstrate the feasibility of this new access method without having to change the Salvager; it may also be useful to run benchmarks even though the changes to the Salvager are not made yet.

COEXISTENCE OF OLD FILES AND NEW FILES

It is desirable that all existing MSF files can still be used even when the new type of file is available. In fact, the MSF type could be used for files that are not very large (a few 256K segments) or for files that are known to be accessed only sequentially such as pl1 listings.

The plan is to give the user a way to express what kind of file he wants to create, and to have vfile capable of dealing with both kinds.

It is also possible to convert an old file into a new file by a trivial transformation.

CONCLUSION

It seems clear to everyone that the MSF technique is not adequate for handling large files such as the IRS files, no matter how much tuning is done with the size of the various page table pools.

The method presented in this document is a reasonable compromise between the conventional buffer manager techniques where the file is not directly accessible by the user, and the current Multics technique, where the entire file can be directly accessible by the user at any given instant. It provides a way for the user to have, at any given instant, some portions of the file directly accessible and to choose, at any given instant, what portions of the file he is interested in making directly accessible.

The basic primitives and the new protocol have been discussed with the vfile expert, since vfile would be the primary user of this facility. It appears that they are adequate and that vfile could be made to work using this protocol, without a major rewriting of the vfile programs.

My recommendation would be to prepare a document describing the implementation to the level of detail required to do a reasonable performance evaluation, and, if satisfactory, to prepare another document describing the amount of work required to get the job done, to the level of detail that would be sufficient to make a reasonable prediction as to how many man-years it would take.